**Experiment Number:** 04

**Aim:** To design and develop Haskell code for given programming problems Part 01.

**Lab Objective:** Design and implement declarative programs in functional and logic programming languages

**Lab Outcome Mapped:** Design and Develop solution based on declarative programming paradigm using functional and logic programming(LO2)

**Requirements:** Any text editor to be able to edit Haskell code and Glasgow Haskell Compiler 8.0+ version.

**Performance:**

**Part 01: Write Haskell function to** Implement safetail function that behaves in the same way as tail, except that safetail maps the empty list to the empty list, whereas tail gives an error in this case. Define safetail using: (a) a conditional expression; (b) guarded equations; (c) pattern matching.

Code Part 01(a)

safet :: [a] -> [a]

saeft xs = if null xs then [] else (tail xs)

main :: IO [Int]

main = do

return (safet [])

Code Part 01(b)

safet (x:xs)

| null xs = []

| otherwise = xs

main :: IO [Int]

main = do

return (safet [])

Code Part 01(c)

safet [] = []

safet (x:xs) = xs

main :: IO [Int]

main = do

return (safet [])

*Note: Students need to copy these codes into three .hs file execute the code and note output. Next change the input by changing* ***return (safet [])*** *to* ***return (safet [<some finite list>])*** *in main function****.***

**Part 02: Write Haskell function to recursive function** to multiply two natural numbers that uses predefined add funion.

add :: Num a => a -> a -> a

add x y = x + y

multiply :: (Ord t, Eq t, Num t) => t -> t -> t

multiply x y

| y == 0 = 0

| y < 0 = (-1) \* (add x (multiply x ((-y)-1)))

| x < 0 = (-1) \* (add (-x) (multiply (-x) (y-1)))

| otherwise = (add x (multiply x (y-1)))

main = do

return (multiply (-4) 5)

*Note: Students need to copy this code into .hs file execute the code and note output. Next change the input by changing* ***return (multiply (-4) 5)*** *to* ***return (****multiply (-4) (-5)****) etc*** *in main function****.***

**Part 03: Write Haskell code** to represent infinite fibobacci series.

fibs = 0 : 1 : zipWith (+) fibs (tail fibs)

main :: IO [Int]

main = return(take 10 fibs)

*Note: Students need to copy this code into .hs file execute the code and note output. Next change the input by changing return (take 123 fibs) or* *return (take n fibs) u can take different value in place of n and re execute to understand code execution.*

**Part 04: Write Haskell code using recursion** to find factorial of a number

factorial x

| x == 0 = 1

| x > 0 = x \* factorial (x-1)

main = do

putStrLn "\nEnter a positive integer: "

m <- getLine

let y = read m :: Int

return(factorial y)

*Note: Students need to copy this code into .hs file execute the code and note output.*

**Conclusion:** Thus we have learned to design and develop programming solutions in haskell.

**Reference:**

[1] Learn you Hakell for great good. A biginner’s guide, <http://learnyouahaskell.com/>, Accessed on 02/12/2020